Parallel Processing

Project 4

turnin_code: pp_p4

Using C/C++, implement a parallel library that provides the user access
to a PGAS system that conforms to the interface described here:

```
    #include "pgas.h"

    return codes:
        PGAS_SUCCESS        successful operation
        PGAS_NO_SPACE       returned by Put if a data item cannot be taken by
                            any server
        PGAS_ERROR          failure (e.g. internal error)

    typedef MPI_Aint PGAS_HANDLE[3];    // created by PGAS by Put
        0 :   server where data is stored
        1 :   address on the server where the data is actually stored
        2 :   size of the stored data

    int PGAS_Init(int max_bytes);
        max_bytes is the maximum number of bytes the PGAS system should
        use on this host before it refuses to take additional data.

    int PGAS_Put(void *buffer, int size, PGAS_HANDLE handle);
        buffer: pointer to the user's data to be stored
```

    size: number of bytes to retrieve from the buffer

    handle: filled in by PGAS; described above

int PGAS_Update(PGAS_HANDLE handle, int offset, int size, void *buffer);

    handle: pgas handle of remote buffer (created via prior Put)

    offset: offset from addr at which to perform the update

    size: number of bytes to update

    addr: addr of data to place into the remote buffer

int PGAS_Get(PGAS_HANDLE handle, int offset, int size, void *buffer);

    handle: obtained from a prior Put

    offset: distance from beginning of buffer (often 0)

    size: num bytes to retrieve from the stored data

    buffer: buffer into which data is to be retrieved

int PGAS_Finalize();

    cleanup and shutdown of the PGAS system (threads etc)

The PGAS library code should run in a separate thread created during PGAS_Init. The library should make sure that it establishes its own communicator for its work.

The makefile should build a library named libpgas.a when the user simply types "make" with no arguments, e.g.:

```
make
```

The makefile should also provide a target that will compile (and link) a C (not C++) program named p4test.c into an executable named p4test. Note that, if your library is implemented in C++, then you must take care to link the C program and C++ library correctly.

The user's programs can be run in this way:

```
    mpiexec -f some_hosts_filename    -n num_ranks    ./p4test    user_pgm_args
```

Use turnin to submit a tar file containing all of your project, including
a makefile that will build the code.   To build, I will type the following:

```
    rm -rf *.o
    make
```

Then I will copy some program to p4test.c and type:

```
    make p4test
```

and run that program several times.

I may build multiple p4test programs for testing.